# SQL Syntax

# Case statement :

The CASE statement in SQL is like a switch-case or if-else ladder — it's used to evaluate conditions and return values accordingly. Let's break it down step by step:

SELECT

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

...

ELSE resultN

END AS alias\_name

FROM table\_name;

SELECT

Marks,

CASE

WHEN Marks >= 90 THEN 'A'

WHEN Marks >= 75 THEN 'B'

WHEN Marks >= 60 THEN 'C'

ELSE 'Fail'

END AS Grade

FROM Students;

## Self Join

SELECT A.column\_name, B.column\_name

FROM table\_name A

JOIN table\_name B

ON A.common\_column = B.common\_column;

**Exemple : select e.name as Employee from Employee e Join Employee m where e.managerId = m.id and  e.salary  > m.salary;**

<https://leetcode.com/problems/employees-earning-more-than-their-managers/>

SELECT     s.product\_id,    fy.first\_year,    s.quantity, s.price FROM Sales s

JOIN (    SELECT       product\_id,         MIN(year) AS first\_year FROM Sales

    GROUP BY     product\_id) fy ON

    s.product\_id = fy.product\_id AND s.year = fy.first\_year;

<https://leetcode.com/problems/product-sales-analysis-iii/>

## Left Outer Join / Left Join

Syntax:

SELECT column\_name(s)   
FROM table1   
LEFT JOIN table2   
ON table1.column\_name = table2.column\_name;

Example:

SELECT Customers.CustomerName, Orders.OrderID   
FROM Customers   
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID   
ORDER BY Customers.CustomerName;

SELECT

'Low Salary' AS category,

SUM(CASE WHEN income < 20000 THEN 1 ELSE 0 END) AS accounts\_count

FROM Accounts

UNION ALL

SELECT

'Average Salary',

SUM(CASE WHEN income BETWEEN 20000 AND 50000 THEN 1 ELSE 0 END)

FROM Accounts

UNION ALL

SELECT

'High Salary',

SUM(CASE WHEN income > 50000 THEN 1 ELSE 0 END)

FROM Accounts;

How can we use a value like 'Low Salary' as a column (category) if it doesn't already exist in the table?”

Let’s clear this up fully — this is a **core SQL concept** involving **literal values** and **aliases**.

How can we use a value like 'Low Salary' as a column (category) if it doesn't already exist in the table?”

Let’s clear this up fully — this is a **core SQL concept** involving **literal values** and **aliases**.

**✅ 1. 'Low Salary' is a literal string, not a column from the table.**

In this part of the query:

sql

CopyEdit

SELECT

'Low Salary' AS category,

...

* 'Low Salary' is a **string value**.
* AS category gives it an **alias**, so in the output, it appears under the column name category.

**✅ 2. Why this works without being in the table?**

SQL lets you **manually create columns** in your result using:

* **Constants/literals** like 'Low Salary', 42, NULL
* **Expressions** like income \* 2, CASE WHEN ...
* Then give them **aliases** using AS to name them.

You're **not referencing a column from the table** — you're **creating** one in the result.

**📌 Example:**

Try this simple query:

sql

CopyEdit

SELECT 'Hello GPT' AS greeting;

Output:

diff

CopyEdit

+------------+

| greeting |

+------------+

| Hello GPT |

+------------+

There’s **no table involved**, but you still get a column greeting with value 'Hello GPT'.

**✅ In Your Case:**

We are building a **custom result** where each row is a salary category and a count:

sql

CopyEdit

SELECT 'Low Salary' AS category, COUNT(...)

It’s like saying:

“Hey SQL, label this row as Low Salary and count how many incomes fall into that.”

**🔁 Summary:**

| **Part** | **Meaning** |
| --- | --- |
| 'Low Salary' | Just a string (like a label) |
| AS category | You're naming the column "category" |
| Not in table? | That’s okay — SQL allows computed or constant columns in output |

Let me know if you want a visual diagram or to try other examples of aliasing like this!

**You said:**

SUM(CASE WHEN income BETWEEN 20000 AND 50000 THEN 1 ELSE 0 END)

**ChatGPT said:**

Great! Let's break down this expression for you:

**🧩 Expression:**

sql

CopyEdit

SUM(CASE WHEN income BETWEEN 20000 AND 50000 THEN 1 ELSE 0 END)

**🔍 What it means — step by step:**

**1. CASE WHEN ... THEN ... ELSE ... END**

This is a **conditional expression**, like an if-else statement.

It reads:

For each row in the Accounts table:

* If income is between **20,000 and 50,000** → return 1
* Else → return 0

So this transforms your data row-by-row into 1s and 0s depending on the condition.

**2. SUM(...)**

This wraps around the CASE to **add up** all the 1s.

Which means:

* You're counting how many rows **satisfy the condition** (income BETWEEN 20000 AND 50000)
* Because every match returns 1, and non-matches return 0.

**✅ Example:**

Say your table has:

text

CopyEdit

income

-------

15000 → ❌ (0)

25000 → ✅ (1)

40000 → ✅ (1)

51000 → ❌ (0)

The CASE returns:

0

1

1

0

Then SUM(...) returns: 1 + 1 = 2

**🧠 Summary:**

| **Expression** | **Meaning** |
| --- | --- |
| CASE WHEN condition THEN 1 ELSE 0 END | Turns true/false into 1/0 |
| SUM(...) | Counts how many times the condition was true |

Top of Form

,

Bottom of Form

## INNER JOIN Syntax:

## SELECT table1.column1, table2.column2, FROM table1

## INNER JOIN table2 ON table1.common\_column = table2.common\_column;

**Exemple: SELECT e.name AS Employee FROM Employee e INNER JOIN Employee m ON e.managerId = m.id WHERE e.salary > m.salary;**

## Window Function: RANK()

Syntax:

RANK() OVER (PARTITION BY column ORDER BY column DESC)

DENSE\_RANK() OVER (PARTITION BY column ORDER BY column DESC)

Exemple: SELECT salary, RANK() OVER (ORDER BY salary DESC) AS rank

FROM Employee;

SELECT salary AS getNthHighestSalary

FROM (

SELECT salary, RANK() OVER (ORDER BY salary DESC) AS salary\_rank

FROM Employee

) AS ranked\_salary

WHERE ranked\_salary.salary\_rank = 2

LIMIT 1;

**Lead () and Lag()**

LEAD(column\_name, offset, default\_value) OVER (

PARTITION BY partition\_column

ORDER BY order\_column

)

LAG(column\_name, offset, default\_value) OVER (

PARTITION BY partition\_column

ORDER BY order\_column

)

| **Parameter** | **Description** |
| --- | --- |
| column\_name | Column to access from the next/previous row |
| offset | (Optional) How many rows forward/backward to look (default = 1) |
| default\_value | (Optional) Value returned if no row exists |

SELECT

salesperson, month, revenue,

LAG(revenue) OVER (PARTITION BY salesperson ORDER BY month) AS prev\_month\_revenue,

LEAD(revenue) OVER (PARTITION BY salesperson ORDER BY month) AS next\_month\_revenue

FROM Sales ;